SISTEM *QUERY* PADA DOKUMEN XML DENGAN MENGGUNAKAN BAHASA SQL

Febriliyan Samopa -- Darlis Heru Murti -- Okhi Oktanio
Program Studi Sistem Informasi, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember
Email : iyan@its-sby.edu
Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember
Email: darlis@its-sby.edu, okhi.oktanio@gmail.com

ABSTRAK

Teknologi XML merupakan teknologi baru yang sangat berguna untuk pertukaran data melalui media apapun dan sistem operasi apapun. Dengan XML, pertukaran data menjadi sangat aman dan mudah untuk dikirimkan dan diambil. Kesemuanya ini dikarenakan bentuk XML yang berupa teks, sehingga memungkinkan untuk dikirim melalui HTTP. Teknologi XML sebenarnya sudah lama diperkenalkan kepada masyarakat, tetapi penggunaannya masih jarang karena sulitnya mengolah data XML.

Kesulitan-kesulitan yang dialami oleh seseorang dalam mengolah dokumen XML yang mereka miliki yaitu mereka harus mempelajari XMLDOM, Xpath, XML Schema dan DTD selain XML itu sendiri. Meskipun semua hal yang tersebut di atas sudah dipelajari, masih ada kesulitan yang lain, yaitu lamanya penelusuran data sebuah XML. Penelusuran data sebuah XML membutuhkan waktu lama karena untuk mendapatkan data tertentu dari sebuah dokumen XML, diperlukan penelusuran satu persatu dari node-node yang dimiliki dokumen XML, sampai menemukan node yang diinginkan.

Pada penelitian ini dikembangkan suatu komponen yang menyerupai class yang terdapat pada ADO. Penggunaan class-class ini akan mempermudah dalam melakukan query, atau manipulasi terhadap dokumen XML, tanpa harus mempelajari terlalu dalam hal-hal yang berhubungan dengan XML. Dengan menggunakan komponen ini, diharapkan seorang pengguna dapat lebih memfokuskan diri pada pengolahan data, tanpa harus mempelajari hal yang baru untuk menerapkan teknologi XML ini.

Dengan menggunakan ekspresi regular dan aturan-aturan tambahan, maka perintah-perintah SQL dirubah menjadi Xpath, dan digunakan untuk mengolah dokumen-dokumen XML. Jika perintah yang didapat adalah perintah untuk memanipulasi sebuah dokumen, maka pada akhir proses akan dilakukan validasi dengan menggunakan dokumen validator dari dokumen XML yang berinteraksi. Jika hasil validasi menunjukkan bahwa dokumen tersebut adalah valid, maka dokumen tersebut disimpan. Sebaliknya jika hasil validasi dokumen menunjukkan bahwa dokumen tidak valid, maka dokumen XML tersebut tidak mengalami perubahan.

***Kata kunci* :** *XML, XML DOM, Xpath, XML Schema, DTD, ADO*

1. PENDAHULUAN

XML adalah bahasa *markup* yang dirancang khusus untuk penyampaian informasi melalui World Wide Web (WWW). XML merupakan suatu format data berbasis teks yang membantu *Developer* dan *Provider* dalam menggambarkan, mengirimkan dan menukarkan data yang terstruktur dari antara berbagai macam aplikasi kepada pengguna untuk keperluan manipulasi. XML juga memberikan fasilitas *transfer* data antar server itu sendiri. XML dapat mengindentifikasikan, menukarkan dan memproses data-data dari berbagai macam *platform* database yang ada dengan tata cara yang general dan format tertentu.

Salah satu cara untuk menginterpretasikan dokumen XML dalam bentuk terstruktur dan terorganisasi dengan menggunakan DOM (Dokumen Object Model). DOM dapat membantu seorang *developer* untuk dapat membuat, memanipulasi, dan me-*load* suatu dokumen XML dalam bentuk *tree* serta mencari kesalahan. DOM terdiri atas dua bagian interface yaitu :

- Core Interfaces
  Core Interfaces terdiri dari hirarki yang mendetail tentang semua koneksi interface yang membentuk kerangka pada XML.
- Flattened Interfaces
  Konsep dasar dari interface ini adalah "Semua adalah Node". Flattened interfaces terdiri dari sekumpulan *interface* yang minim.

Dari pengertian di atas terlihat bahwa seorang pengguna awal yang masuk dan menggunakan XML sebagai alat penyampaian informasi akan mengalami kesulitan dalam mengakses data XML tersebut. Ini semua dikarenakan kerumitan DOM sendiri yang memilki kompleksitas dan kerumitan yang tinggi bagi pengguna awal.

Apabila menggunakan XML DOM (MSXML atau dalam class .NET terdapat namaspace System.Xml) untuk mengakses data pada XML, proses

mendapatkan harus menelusuri node per node dari data XML tersebut. Cara ini memiliki kompleksitas waktu yang sangat tinggi sehingga tidak bisa dikatakan efisien apabila seorang *user* hanya memerlukan data tertentu dari dokumen XML yang *user* tersebut miliki. Cara yang lebih efisien apabila ingin melakukan *query* adalah menggunakan Xpath.
Xpath merupakan bahasa standar yang merupakan standar W3C untuk melakukan *query* pada dokumen XML. Struktur Xpath memiliki struktur yang sama dengan path-path yang selama ini telah dikenal sebagai contoh bisa dilihat pada Gambar 1. dan contoh Xpath yang dapat di lihat pada Gambar 2. Dengan Xpath ini seorang *user* bisa melakukan *query* dan memanipulasi data yang diinginkan.

```
http://gmail.google.com/gmail/help/start.html
http://www.amazon.co.uk/exec/obidos/ASIN/1902699629/ref=nosim
C:\Program Files\Ahead
```

Gambar 1. Regular Path

```
/catalog/cd/*
/catalog/cd/price
Data/users/user
```

Gambar 2. Xpath

Teknologi yang baru ini mengharuskan setiap *user* untuk mempelajari 4 hal baru bila ingin menggunakan XML, yaitu XML, XML DOM, Xpath dan tentunya DTD atau XML schema bila menginginkan validasi terhadap dokumen XML. Dalam penelitian ini dibentuk sebuah komponen seperti ADO (Connection dan Recordset) yang memungkinkan seorang *user* dapat melakukan *query* dan manipulasi data tanpa harus mempelajari hal tersebut terlalu dalam.

2. XML

eXtensible Markup Language (XML) merupakan *subset* dari Standard *Generalized Markup Language* (SGML) yang ditetapkan oleh *World Wide Web Consortium* (W3C). XML bisa dikatakan bentuk dari SGML yang lebih terbatas.
XML merupakan suatu format data berbasis teks yang membantu *developer* atau *provider* dalam menggambarkan, mengirimkan dan menukar data yang terstruktur dari antara berbagai macam aplikasi kepada pengguna untuk keperluan manipulasi atau presentasi. XML dapat mengidentifikasi, menukarkan dan memproses data-data dari berbagai macam platform database maupun sistem operasi yang ada dengan tata cara yang general.
XML memiliki kemiripan dengan HTML tetapi HTML memiliki *tag-tag* yang pasti seperti Gambar 3. sedangkan XML memiliki fleksibilitas yang tinggi dalam strukturnya seperti Gambar 4. XML merupakan pelengkap dari HTML, bukan merupakan pengganti dari HTML tersebut, karena XML lebih mengarah pada bentuk data sedangkan HTML mengarah pada bagaimana data tersebut ditampilkan. XML memiliki banyak keuntungan yaitu :

- Ekstensibilitas : Dapat menentukan *tag-tag* sesuai dengan kebutuhan.
- Dapat dijadikan alat pertukaran data dari sistem yang berbeda.
- XML merupakan *file* teks sehingga bisa dilakukan pertukaran data melalui internet (HTTP).
- Dengan satu *file* XML dapat dilakukan manipulasi tampilan sesuai dengan keinginan pengguna.
- Proses *query* dari data XML lebih cepat karena bentuk struktur yang berupa *tree*.

```
E bgColor=#ffffff cellPadding=0 cellSpacing=0 class=right
th="100%">
TR>
<TD align=middle class=right><BR><A href="http://www.net2.co.uk/"
 target=_blank>Buy UK Domain Names</A><BR><A
 href="http://www.123domainnames.co.uk/" target=_blank>Register
 Domain Names</A><BR><BR></TD>
/TR>
E>
```

Gambar 3. Dokumen HTML

```
<bookstore>
  <book genre='novel' ISBN='10-861003-324'>
    <title>The Handmaid's Tale</title>
    <price>19.95</price>
  </book>
  <book genre='novel' ISBN='1-861001-57-5'>
    <title>Pride And Prejudice</title>
    <price>24.95</price>
  </book>
</bookstore>
```

Gambar 4. Dokumen XML

3. DTD

Document Type Declaration (DTD) digunakan untuk mendefinisi sebuah dokumen XML yang legal (*valid*), mulai dari struktur dan type datanya. DTD dapat di-*include*-kan secara langsung ke dalam dokumen XML atau berdiri sendiri sebagai *file* eksternal.

DTD ini sifatnya optional, suatu dokumen XML yang *well-formed* tidak harus menyertakan DTD pada setiap dokumen yang dibuat, tetapi bila menyertakan maka dokumen XML ini disebut *valid* bila mengikuti semua aturan yang telah ditulis di dalam DTD.

Pada dasarnya DTD dapat dikategorikan menjadi 2 macam, yaitu :

- Internal DOCTYPE declaration
  DTD di-*include*-kan kedalam XML *file* sehingga harus dideklarasikan di dalam *tag* DOCTYPE. Contoh sintaks :

<!DOCTYPE root-element [element-declaration]>

Untuk melihat lebih jelasnya bisa dilihat pada gambar 5.

- External DOCTYPE declaration
  DTD dibuat pada *file* yang terpisah sehingga sintaks DOCTYPE harus menggunakan :

  <DOCTYPE root-element SYSTEM "*file*name">

  Untuk lebih jelasnya dapat dilihat pada gambar 6.

```
<?xml version="1.0"?>
<!DOCTYPE note [
  <!ELEMENT note (to,from,heading,body)>
  <!ELEMENT to     (#PCDATA)>
  <!ELEMENT from   (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body   (#PCDATA)>
]>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend</body>
</note>
```

Gambar 5. Dokumen XML dengan Internal DOCTYPE declaration

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Gambar 6. Dokumen XML dengan Eksternal DOCTYPE Declaration

4. XSD

XML Schema Definition (XSD) merupakan suatu alternatif dari validator dokumen XML selain DTD. XSD lahir karena ketidakpuasan pengguna dengan kemampuan DTD dan kesulitannya. Adapun kelemahan DTD adalah :

- DTD menggunakan bahasa yang berbeda dengan XML sehingga seorang pengguna harus mempelajari lagi bahasa yang digunakan oleh DTD.
- DTD memiliki tipe data yang terbatas (10 tipe data)
- Tidak mendukung XML namespace

Karena ketiga kelemahan inilah maka XML *schema* atau disebut juga XSD mulai dibuat.
XSD memiliki banyak kemampuan sehingga XSD mempunyai masa depan yang lebih baik dibandingkan dengan DTD. Dengan XSD seorang *user* dapat mendefinisikan :

- Elemen yang diijinkan muncul pada dokumen XML
- Atribut yang diijinkan muncul pada dokumen XML
- Elemen mana yang merupakan *child-element*
- Urutan dari child-element
- Jumlah dari child-element
- Tipe data dari sebuah elemen atau atribut
- Mengarah pada data yang *object oriented*
- Lebih mudah bekerja dari data sebuah database
- Pola dari suatu data

XSD diterbitkan oleh *Microsoft* tetapi kemudian menjadi standar W3C pada bulan Mei 2001. Dengan XSD diharapkan pertukaran data semakin lebih mudah, stabil, aman dan luas. Contoh perbandingan serta perubahan DTD dan XSD dapat dilihat dari Gambar 7 dan Gambar 8.

```
<!ELEMENT BookStore (Book+)>
<!ELEMENT Book (Title, Author, Date,
                ISBN, Publisher)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>
```

Gambar 7. Contoh DTD

Gambar 8. Contoh XSD

5. XPATH

Bahasa *Xpath* adalah suatu aturan sintaks untuk mendefinisikan bagian dari sebuah dokumen XML. Xpath menggunakan *path* untuk mendefinisikankan elemen XML, *path* yang digunakan memiliki kemiripan dengan *path* yang biasa digunakan. Xpath merupakan standar yang ditetapkan oleh W3C.
Path yang digunakan oleh Xpath sangat mirip dengan *path* dari suatu *file*, untuk dapat membandingkan akan diberikan contoh seperti di bawah ini.

File path : www.bekas.com/hardware/main.asp
Xpath : /catalog/cdcatalog/title

6. System.Xml dalam .NET

Namespace System.Xml memiliki sekumpulan *class-class* yang mendukung standarisasi dalam memproses dokumen XML. Standar yang di dukung adalah ;

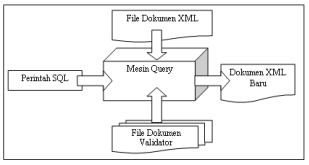
- XML 1.0 - http://www.w3.org/TR/1998/REC-xml-1998021 (termasuk mendukung DTD)
- XML Namespaces - http://www.w3.org/TR/REC-xml-names/ (stream level dan DOM)
- XSD Schemas - http://www.w3.org/2001/XMLSchema
- XPath expressions - http://www.w3.org/TR/xpath
- XSLT transformations - http://www.w3.org/TR/xslt
- DOM Level 1 Core - http://www.w3.org/TR/REC-DOM-Level-1/
- DOM Level 2 Core - http://www.w3.org/TR/DOM-Level-2/

7. REGULAR EXPRESSION

Regular Expression (ekspresi regular, regexp, regex, RE) adalah sebuah bahasa mini untuk mendiskripsikan *string* atau teks. Regex dapat dipakai untuk mencocokkan sebuah *string* dengan sebuah pola. Analogikan hal ini dengan fungsi *string* yang sudah dikenal seperti substring(), strcmp() atau strreplace(), tetapi regex lebih ampuh dan memiliki kelebihan lain dibandingkan fungsi-fungsi sederhana dari *string*.
Regex sebetulnya telah diciptakan lama dan telah dipakai program dan bahasa pemrograman, meskipun mula-mulanya hanya dijumpai dalam beberapa versi grep di unix dan DOS. Sekarang regex digunakan oleh banyak aplikasi karena keampuhannya, keringkasannya dan kecepatannya.
Keampuhan regex dapat dilihat dimana regex dapat dengan mudah mengekstrak pola *string* untuk diambil angka, nomor telepon, URL dan lainnya. Keringkasannya membuat regex dapat menggantikan belasan bahkan puluhan baris kode program hanya dengan sebuah pola. Kecepatannya membuat seorang pengguna terhindar dari waktu yang terbuang percuma karena membandingkan puluhan atau bahkan ratusan *string* untuk mendapatkan data yang diinginkan.

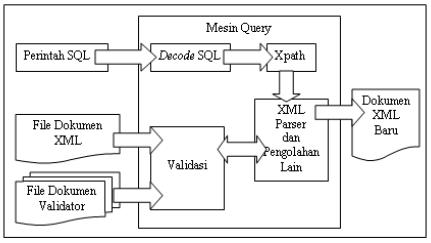
8. PERANCANGAN SISTEM

Sistem ini menerima tiga *input*, yaitu perintah SQL, dokumen XML, dan dokumen validator (DTD atau XSD). Setelah diolah akan dihasilkan suatu *output* berupa dokumen XML hasil manipulasi atau dokemen XML baru hasil *query*. Untuk mengetahui lebih lanjut dapat dilihat pada Gambar 9.



Gambar 9. Bentuk *input* dan *output* sistem

Di dalam mesin *query* inilah semua pengolahan terjadi sehingga menghasilkan dokumen XML baru.
Rancangan Mesin *query* yang akan mengolah data-data seperti pada Gambar 9 dapat dilihat pada Gambar 10 di bawah ini.

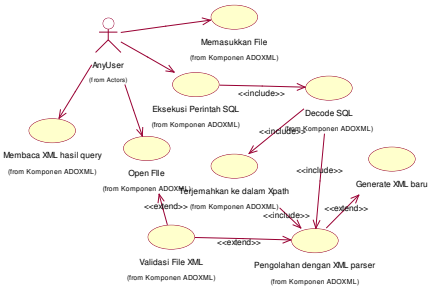


Gambar 10. Rancangan sistem (mesin *query*)

Mesin *query* melakukan *decode* SQL karena XML *parser* yang standar tidak menyediakan *parsing* XML dengan menggunakan bahasa SQL.
Setelah berhasil melakukan *decoding* perintah tersebut dirubah menjadi Xpath, yang merupakan bahasa standar *query* dokumen XML, kemudian digunakan XML *parser* untuk mengolah dokumen XML yang telah menjadi *input* pada mesin *query* ini.
Dokumen XML dan dokumen validator yang menjadi *input* divalidasi terlebih dahulu sebelum diolah. Setelah selesai divalidasi baru kemudian dokumen tersebut diolah oleh XML *parser* dan menghasilkan dokumen baru sebagai hasilnya.

9. PERANCANGAN KOMPONEN

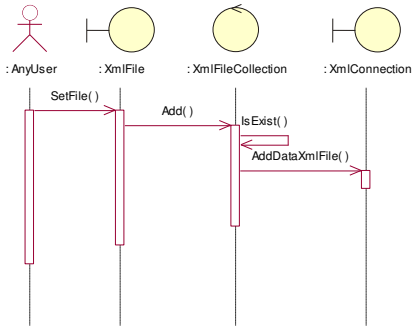
Pada bagian ini akan dijelaskan tentang pemodelan komponen yang merupakan bentuk dari mesin *query* yang berguna untuk memanipulasi dan melakukan *query file* dokumen XML. Komponen ini diberinama ADOXML karena komponen ini dibentuk mirip dengan ADO yang selama ini dikenal.
Komponen ini mempunyai satu aktor, yaitu Any*User* karena siapapun dapat menggunakan komponen ini tanpa hak-hak khusus dalam menggunakan komponen ini.
Sistem ini hanya mempunyai satu aktor karena komponen ini dapat digunakan siapapun tanpa memberi hak khusus kepada setiap *user*. Untuk mengetahui lebih jelas bagaimana alur dari pemodelan sistem ini maka digambarkan dengan menggunakan alur *use case* seperti pada gambar 11.



Gambar 11. *Use case* pada komponen

10. MEMASUKKAN *FILE*

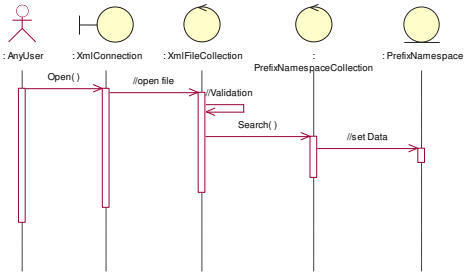
Any*User* memasukkan *path file* XML, semua *path file* untuk validasi (XSD/DTD) dan memberikan sebuah alias untuk *file* XML yang ingin diolah datanya.
Kemudian *control* Xml*File*Collection melakukan pengecekkan apakah sudah ada *file* tersebut dengan alias sebagai *unique id*-nya. Jika tidak ada maka data tersebut disimpan pada XmlConnection. Untuk melihat alur prosesnya dapat dilihat pada Gambar 12.



Gambar 12. *Sequence diagram* memasukkan *file*

11. OPEN *FILE*

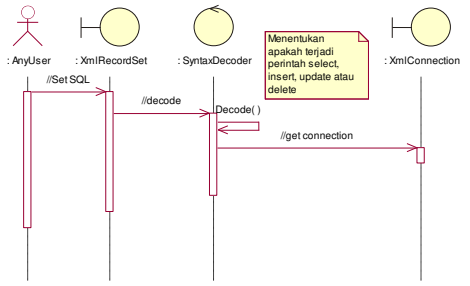
Pada bagian ini dijelaskan bagaimana *file* yang telah dimasukkan dibuka dan divalidasi. Aktor Any*User* melakukan perintah untuk membuka *file* yang telah dimasukkan pada *boundary* XmlConnection kemudian *control* Xml*File*Collection akan membuka *file* XML dan semua *file* validator.
Setelah berhasil membuka kemudian *control* Xml*File*Collection akan melakukan validasi *file* XML terhadap semua *file* validatornnya. Jika valid maka *control* Xml*File* akan memerintahkan *control* PrefixNamespaceCollection untuk pencarian *prefix* dan *namespace* yang ada pada *file* XML dan bila terdapat *namespace* yang tidak mempunyai *prefix* maka secara otomatis akan diciptakan *prefix*. Kemudian *prefix* dan *namespace* tersebut disimpan pada *entity PrefixNamespace*. *Sequence diagram* proses ini dapat dilihat pada Gambar 13.



Gambar 13. Sequence diagram open file

12. EKSEKUSI PERINTAH SQL

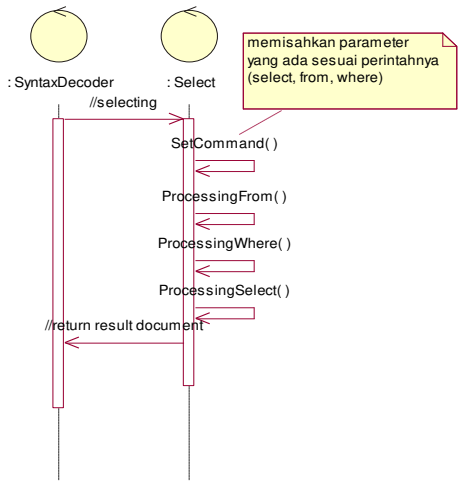
Setelah *file* yang diinginkan telah terbuka maka Any*User* dapat melakukan perintah untuk memanipulasi dan melakukan *query* data dokumen XML dengan menggunakan bahasa SQL.
Any*User* dapat memberikan input berupa *multicommand* (pemecahan *multicommand* dapat dilihat pada Gambar 3.11) SQL yang kemudian oleh *control* SyntaxDecoder diproses untuk menentukan apakah perintah itu masuk ke dalam perintah *query* (*selecting)* atau manipulasi (*insert, update* dan *delete*).
Setelah melakukan *decode control* ini mengambil data *file* XML yang telah dibuka di *class* XmlConnection. Untuk mengetahui lebih jelas dapat dilihat *sequence diagram* seperti pada Gambar 14.



Gambar 14. *Sequence diagram* eksekusi SQL

13. SELECT

Jika *control* SyntaxDecoder mendapatkan bahwa perintah yang dimasukkan *user* adalah perintah *select* maka *control* ini menciptakan objek baru (*control* Select) dan memerintahkan kepada objek baru ini untuk melakukan *query* atau *selecting*.
Setelah itu *control* Select melakukan pemisahan parameter yang ada pada perintah SQL tersebut menjadi 3, yaitu parameter *select*, *from* dan *where*. Selesai memisahkan parameter tersebut *control* ini memprosesnya satu per satu dan setelah selesai *control* ini memberikan nilai kembalian berupa dokumen XML yang baru . Untuk mengetahui alur pada proses ini dapat dilihat pada Gambar 15.



Gambar 15. Sequence diagram proses selec*t*

Parameter *from* diproses dengan dua pilihan, yaitu jika data yang didapat adalah perintah *select* maka *control* Select akan memerintahkan *control* SyntaxDecoder untuk melakukan Decode terhadap perintah ini dan menciptakan objek *control* Select baru untuk mengolahnya, sedangkan jika data yang didapat merupakan *file* dokumen XML, data tersebut langsung diolah.

Sebelum melakukan pengolahan, *control* ini menciptakan dokumen XML baru sebagai hasil dari proses ini. Kemudian *control* select melakukan perubahan terhadap *prefix* yang ada dalam dokumen ini untuk menghindari penggunaan *prefix* yang sama pada dokumen-dokumen XML yang berinteraksi.

Proses pencarian dan pembentukan *prefix* ini dilakukan oleh *control* NewPrefixNamespaceCo dan kemudian menyimpannya pada *entity* NewPrefixNamespace.

Setelah selesai melakukan semuanya itu, dilakukan perubahan parameter *from* menjadi perintah Xpath yang kemudian Xpath ini melakukan *query* pada masing-masing dokumen XML dan menyimpannya pada dokumen XML baru yang telah diciptakan. Cara ini juga dilakukan untuk parameter yang lain.

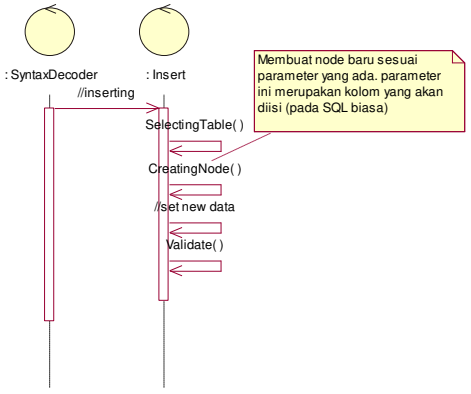
14. INSERT, UPDATE, DELETE

Jika *control* SyntaxDecoder mendapatkan bahwa perintah yang dimasukkan oleh *user* adalah perintah *insert* maka *control* SyntaxDecoder akan membentuk objek baru yaitu *control* Insert dan memerintahkan *control* Insert ini untuk melakukan *inserting*.

Pada *control* Insert parameter yang ada setelah sintaks *insert into* (parameter nama tabel pada sintaks SQL biasa) digunakan untuk parameter pada Xpath sebagai alat untuk melakukan *query* pada dokumen XML yang dituju.

Setelah selesai melakukan *query control* ini melakukan pembuatan *node* baru sesuai parameter kolom dan kemudian menyimpan data yang baru pada *node* tersebut.

Setelah semuanya selesai, hal terakhir yang dilakukan adalah melakukan validasi terhadap validator dari dokumen XML yang berinteraksi. Jika tidak *valid* maka semua proses *inserting* dibatalkan. S*equence diagram* dari proses ini dapat dilihat pada Gambar 16.



Gambar 16. Sequence diagram proses insert

Jika *control* SyntaxDecoder mendapatkan bahwa perintah yang dimasukkan oleh *user* adalah perintah *update* maka *control* SyntaxDecoder akan membentuk objek baru yaitu *control* Update dan memerintahkan *control* Update ini untuk melakukan *updating*.

Pada *control* Update parameter yang ada setelah sintaks *update* (parameter nama tabel pada sintaks SQL biasa) dan sintaks *where* digunakan sebagai parameter pada Xpath dalam melakukan *query* pada dokumen XML yang dituju.

Setelah selesai melakukan *query control* ini melakukan penggantian data sesuai dengan parameter nilai yang menjadi masukkan dari *user*.

Setelah semuanya selesai, hal terakhir yang dilakukan adalah melakukan validasi terhadap validator dari dokumen XML yang berinteraksi. Jika tidak valid maka semua proses *updating* dibatalkan. *Sequence diagram* dari proses ini dapat dilihat pada Gambar 17.

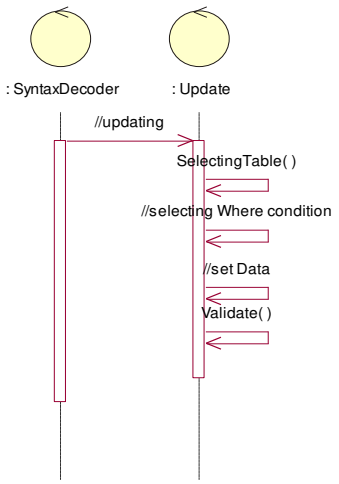
Jika *control* SyntaxDecoder mendapatkan bahwa perintah yang dimasukkan oleh *user* adalah perintah *delete* maka *control* SyntaxDecoder akan membentuk objek baru yaitu *control* Delete dan memerintahkan control Delete ini untuk melakukan *deleting*.

Pada *control* Delete, parameter yang ada setelah sintaks *delete from* (parameter nama tabel pada sintaks SQL biasa) dan sintaks *where* digunakan sebagai parameter pada Xpath dalam melakukan *query* pada dokumen XML yang dituju.

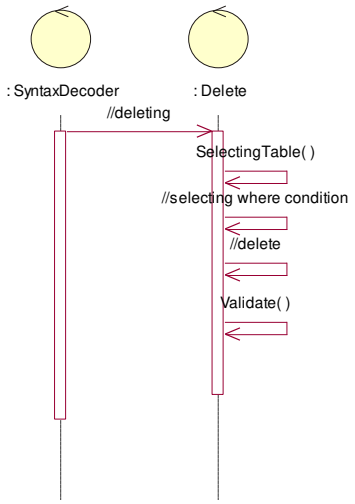
Setelah selesai melakukan *query control* ini melakukan penghapusan *node* yang terpilih dari proses *query* Xpath.

Setelah semuanya selesai, hal terakhir yang dilakukan adalah melakukan validasi terhadap validator dari dokumen XML yang berinteraksi. Jika tidak valid

maka semua proses *deleting* dibatalkan. *Sequence diagram* dari proses ini terdapat pada Gambar 18.



Gambar 17. Sequence diagram proses update



Gambar 18. Sequence diagram proses delete

15. UJI COBA DAN ANALISIS

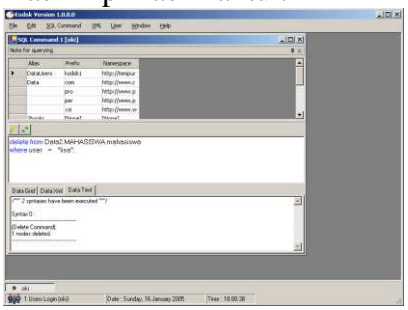
Dokumen XML yang dipakai untuk melakukan uji coba ada 4 dokumen XML yaitu FRS.xml dengan alias FRS , Mhs.xml dengan alias Mhs, Data2.xml dengan alias Data2 dan Data*User*s.xml dengan alias Data*User*s.

Pada Gambar 19 adalah contoh dan hasil eksekusi perintah manipulasi yaitu perintah *delete*

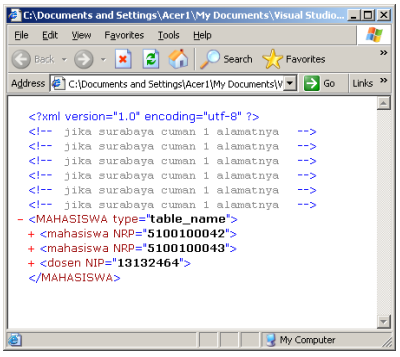
SQL :

```
delete from
Data2.MAHASISWA.mahasiswa
where user    =    "lisa";
```

Hasil Aplikasi Bantu :



Hasil Dokumen XML :



Gambar 19. Eksekusi perintah *delete*

Uji coba pada perintah *select* memiliki kompleksitas yang tinggi karena melibatkan banyak *file* dokumen XML. Berikut ini uji coba perintah *select* yang terdiri dari 3 dokumen XML yang telah disebutkan pada bagian awal bab ini. Uji coba ini akan memberikan nilai kembalian berupa dokumen XML dalam bentuk objek dari *class* XmlDocumen dan *class* DataSet. Contoh eksekusi perintah *select* dapat dilihat pada Gambar 20.

Evaluasi dari hasil uji coba yang telah dilaksanakan adalah sebagai berikut :

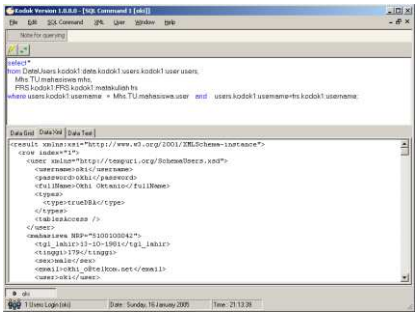
- Pada saat melakukan *query* sering terjadi kebingungan penggunaan *prefix* yang sama pada dokumen XML yang berinteraksi.
- Meskipun komponen ini memudahkan seorang *user* tetapi seorang *user* masih harus mengetahui tentang penggunaan *prefix* dan *namespace* jika *user* ini ingin memanipulasi dokumen XML yang memiliki *namespace* pada dokumennya.
- Pada umumnya semua aplikasi berjalan dengan baik karena uji coba yang dilakukan sudah memberikan hasil yang diinginkan.

SQL :

```
select *
from
DataUsers.kodok1:data.kodok1:users
.kodok1:user users,
     Mhs.TU.mahasiswa mhs,
FRS.kodok1:FRS.kodok1:matakuliah
frs
where users.kodok1:username    =
Mhs.TU.mahasiswa.user     and
users.kodok1:username=frs.kodok1:u
sername;
```

Hasil Aplikasi Bantu :

Hasil Dokumen XML :



Gambar 20. Eksekusi perintah *select*

16. SIMPULAN DAN SARAN

Simpulan sebagai hasil penelitian ini adalah sebagai berikut :

a. Melakukan *parsing* perintah SQL dapat dilakukan dengan menggunakan ekspresi regular yang berisikan setiap aturan dari bahasa SQL. Ekspresi regular ini akan menangkap semua parameter dari perintah SQL dan menjadikannya beberapa *group* yang nantinya digunakan untuk melakukan *query* dengan Xpath. Setelah mendapatkan parameter yang dibutuhkan, parameter ini dirubah menjadi *path* yang merupakan standar dari Xpath.
b. Penggunaan ekspresi regular untuk melakukan *parsing* dirasakan kurang cukup karena ekspresi regular tidak dapat menangkap *error* atau perintah SQL yang tidak sesuai dengan aturan. Sehingga penggunaan validasi untuk hasil yang ditangkap oleh ekspresi regular sangat diperlukan.
c. Setiap data yang berhasil dimanipulasi tidak secara otomatis melakukan validasi terhadap validatornya, sehingga dibuat suatu aturan yang mengharuskan proses manupilasi untuk melakukan validasi ketika selesai melakukan manipulasi. Informasi menggabungkan validator yang satu dengan yang lainnya dilakukan oleh validator yang bersangkutan.
d. *Error* yang dilakukan oleh *user* karena kesalahan data ditangkap oleh sistem sedangkan *error* karena kesalahan perintah SQL ditangkap oleh ekspresi regular dan aturan tambahan yang melengkapi ekspresi regular.

Untuk pengembangan lebih lanjut dari penelitian ini dapat dilakukan :

a. Penambahan kemampuan dari komponen sehingga dapat melakukan perintah-perintah SQL yang lain.
b. Membuat aplikasi yang dapat menyimpan data dengan format XML.

17. DAFTAR PUSTAKA

1. Costello, Roger L, *XML Schema*, http://xfront.com, 2002
2. Donald, Ros, *Regular Expression Designer*, RadSoftware, 2003
3. Haryanto, Steven, *Regex*, Dian Rakyat, 2004
4. Kurniawan, Agus, dkk, *Pengenalan Bahasa C#*, Project Otak, Indonesia.net, 2004
5. *Microsoft Developer Network*, Microsoft Corporation, 2004
6. O'Reilly, *XML*, http://www.xml.com, 2002
7. *Regex Library*, http://www.regexlib.com
8. *Xpath,* http://www.w3schools.com